

21

QUE LE FORTH SOIT AVEC VOUS

FEVRIER 1986

VOUS AVEZ LE TICKET CHIC, NOUS AVONS LE FORTH CHOC.

SORTIE

CORRESPONDANCE

RATP

Une autre musique

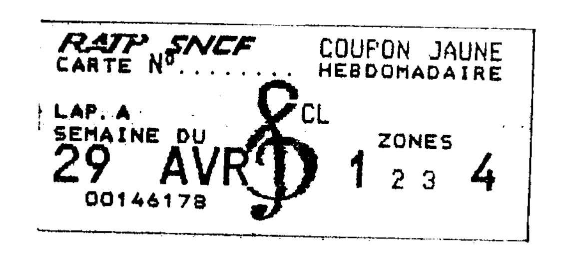
EDITORIAL

L'année 1986 commence mal pour la planète, car entre les guerres, les attentats, les dictateurs déchus et les PADAK (Paris-Dakar) qui capotent, il semble difficile de faire un scoop.

En effet, qui croirait qu'une revue technique puisse attirer l'attention avec un article, à fortiori (sans jeu de mot) concernant le langage FORTH. Or, c'est un scoop que JEDI vous propose ce mois-ci, en dévoilant en avant-première les dessous d'une manifestation culturelle dont l'impact (nous le souhaitons) sera du même ordre que l'emballage du Pont Neuf par Christo.

Par cet article, JEDI démontre qu'une revue de très faible tirage, disposant de moyens dérisoires sait aussi faire de l'information.

Puisse l'esprit JEDI continuer à se répandre, avec votre aide et vos suggestions. Car, à coté des grands pouvoirs médiatiques de la radio, la télévision et la presse à grand tirage, les revues d'association ont leur place à tenir et doivent être prises en considération.



SOMMAIRE

ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS
Tel: (1) 45.42.88.90 (de 10h à 18h)

FORTH MICRO TRAITEMENT DE TEXTE

par Allan HANSEN

> Parmis les divers documents inter-clubs nous arrivant des U.S.A. (ORANGE FORTH), le programme qui suit a particulièrement attiré notre attention. En effet, il permet de réaliser un véritable traitement de texte avec très peu de moyens. Nous vous livrons ce document tel que nous l'avons reçu.

QUICK TEXT FORMATTER
by
Leo Brodie, (FORTH DIMENSIONS, IV,3.)

Westminster,01/10/85

FORTH INTEREST GROUP
Orange County

As you can see, to write a letter with this word processor is to write a FORTH program. Having this system sitting on top of FORTH allows the full power of FORTH to be called upon while writing a document or letter.
To illustrate this, let us redefine the meaning of pp (new paragraph):

SPECIAL EFFECTS using existing FORTH words are easy to include. I plan to implement a word which prints a letterhead using the graphics capability of the EPSON printer.
Enjoy,
Allan Hansen

21:55:34 01/10/85

```
SCR # 160
PR.ON COMPRESS.OFF start center[ QUICK TEXT FORMATTER] cr center
[ by] cr center[ Leo Brodie, (FORTH DIMENSIONS, IV,3.)] 3 crs [
Westminster,] .DATE 3 crs [ FORTH INTEREST GROUP] cr [ Orange Co
unty] 3 crs [ As you can see, to write a letter with this word p
rocessor is to write a FORTH program. Having this system sitting
 on top of FORTH allows the full power of FORTH to be called upo
n while writing a document or letter.] pp [ To illustrate this,
let us redefine the meaning of pp ( new paragraph ):] PR.OFF DEF
INITIONS : pp 2 crs ; PR.ON pp ITALICS.ON [ SPECIAL EFFECTS] ITA
LICS.OFF [  using existing FORTH words are easy to include.  I p
lan to implement a word which prints a letterhead using the grap
hics capability of the EPSON printer.] FORGET pp pp ( This comme
nt is ignored, because it is parsed by FORTH) [ Enjoy, ] cr [ Al
lan Hansen] PR.OFF end
```

20:39:25 01/10/85

```
SCR # 10
( TEXT FORMATTER )                                      ( ABH 011085 )
 SWP DEFINITIONS
 78 VARIABLE PAPER ( WIDTH )
 10 VARIABLE LMARGIN PAPER @ 10 - VARIABLE RMARGIN
  6 VARIABLE TMARGIN          55 VARIABLE BMARGIN
  0 VARIABLE DELIMITER         0 VARIABLE XTRA
  0 VARIABLE ACROSS            0 VARIABLE DOWNWARD
  1 VARIABLE VSPACE            1 CONSTANT single
  2 CONSTANT double            0 VARIABLE page#
: SKIP ( N -- ) ( MOVE N SPACES )
  DUP ACROSS +! SPACES ;
: \line ( -- ) ( GO TO NEXT LINE )
  0 ACROSS ! CR 1 DOWNWARD +! LMARGIN @ XTRA @ + SKIP ;
-->
```

```
20:39:45 01/10/85
SCR # 11
( TEXT FORMATTER )                                      ( ABH 011085 )
: start ( BEGIN NEW DOCUMENT )
 O DOWNWARD ! TMARGIN @ O DO \line LOOP
 1 page# ! ;
: newpage ( BEGIN NEXT PAGE )
 1 page# +!
 CR FF O DOWNWARD ! 65 SPACES page# @ 4 .R
 TMARGIN @ O DO \line LOOP ;
: spacing ( N -- ) ( CAUSE MULTIPLE SPACING )
 VSPACE ! ;
: cr ( NEW LINE, IF NECESSARY NEW PAGE )
 VSPACE @ O DO
 DOWNWARD @ BMARGIN @ > IF newpage ELSE \line THEN
 LOOP ;
-->

20:40:02 01/10/85
SCR # 12
( TEXT FORMATTER )                                      ( ABH 011085 )
: crs ( N -- ) ( N cr'S IN SEQUENCE )
 O DO cr LOOP ;
: tab ( N -- ) ( TABULATE RELATIVE TO LEFT MARGIN )
 ACROSS @ LMARGIN @ - - 1 MAX SKIP ;
: indent ( N -- ) ( TAB N IN THIS AND FOLLOWING LINES )
 DUP XTRA ! tab ;
: pp ( PARAGRAPH )
 cr 5 SKIP ;
-->

20:40:17 01/10/85
SCR # 13
( TEXT FORMATTER )                                      ( ABH 011085 )
: hang-ind ( N -- ) ( TAB N IN THIS AND TAB N+3 IN FOLLOWING )
 DUP tab 3 XTRA ! ;
 : ?NEAR ( -- F ) ( TEST IF NEAR RIGHT MARGIN )
  ACROSS @ RMARGIN @ > ;
 : ?WRAP ( -- F ) ( TEST IF AT RIGHT EDGE OF PAPER )
  ACROSS @ PAPER @ = ;
 : LETTER ( -- C ) ( GET CURRENT LETTER FROM BUFFER )
  BLK @ BLOCK IN @ + C@ ;
 : FLUSH-LEFT ( AFTER cr, DO NOT OUTPUT A BL )
  cr LETTER BL = IF 1 IN +! THEN ;
 : PARSE ( C -- ) ( DISPLAY TEXT TO DELIMITER C )
  DELIMITER ! BEGIN LETTER 1 IN +! DUP
  DELIMITER @ = IN 1023 = OR NOT WHILE DUP EMIT
  1 ACROSS +! BL = ?WRAP OR IF ?NEAR IF FLUSH-LEFT
  THEN THEN REPEAT DROP O XTRA ! ; -->

20:40:37 01/10/85
SCR # 14
( TEXT FORMATTER )                                      ( ABH 011085 )
: [ ( PARSE TEXT UNTIL ] )
 ASCII ] PARSE ;
: sub ( SUBSECTION )
 cr 5 hang-ind ;
: subsub ( SUBSUBSECTION )
 cr 10 hang-ind ;
: center[ ( CENTER TEXT UNTIL ] )
 IN @ RMARGIN @ LMARGIN @ -
 5 + ASCII ] WORD HERE C@ - 2 / tab IN ! [ ;
: r[ ( N -- ) ( RIGHT JUSTIFY IN FIELD N WIDE )
 IN @ SWAP ASCII ] WORD HERE C@ - O MAX SKIP IN ! [ ;
: load ( SCREEN -- ) ( LOAD A SCREEN )
 LOAD ;
-->

20:40:54 01/10/85
SCR # 15
( TEXT FORMATTER )                                      ( ABH 011085 )
CODE end ( -- ) ( END OF DOCUMENT )
 PLA, IP STA, PLA, IP 1 + STA, NEXT JMP, END-CODE IMMEDIATE
 ;S
```

Forth + assembleur 6502

SCULPTURES SONORES EN FORTH

***En bref** : un réseau de cent micro-ordinateurs, tournant 24 heures sur 24 dans un lieu public... un éditeur graphique-couleur à télécommande radio... un ensemble multi-processeurs, multi-tâches, temps réel...*
Une application Forth ambitieuse pour le plaisir de vos oreilles.

Au printemps 1986, si vous empruntez les deux cent mètres de tapis roulants qui relient la gare Montparnasse à la station de métro du même nom, votre oreille percevra peut-être autre chose que le brouhaha incessant de la foule ou les habituels rythmes africains... Si vous ne vous souvenez pas de cet article, vos yeux chercheront en vain à localiser une source sonore, mais ne rencontreront que le regard perplexe d'autres personnes qui, comme vous, commençaient à se demander si leurs sens ne les trahissaient pas. Non, la RATP n'a pas poussé le chic jusqu'à inviter Dieu pour distraire ses usagers : ce sont deux cent haut-parleurs (un tous les deux mètres de chaque coté, astucieusement dissimulés) qui vous chuchotent une sculpture sonore de Max Neuhaus, un artiste américain qui rêve de ce projet depuis son premier passage en ce lieu en 1973 ; dix ans d'efforts lui ont permis de convaincre la RATP et d'autres sponsors pour financer ce projet.

Une sculpture sonore, ça ne peut ni s'écrire sur une partition, ni s'enregistrer : ce n'est pas une séquence de sons dans le temps, mais une topologie sonore, que l'artiste adapte spécialement aux caractéristiques acoustiques du lieu ; les sons se déplacent, interfèrent, se réfléchissent sur les parois, le tout presque au seuil de l'audibilité. Les conceptions acoustiques de Max Neuhaus sont des plus intéressantes pour leur originalité (voir les articles parus dans la presse : International Herald Tribune 21/1/1983, Le Monde 15-16/5/1983), mais c'est l'aspect technique du projet qui nous intéresse ici.

LE MATERIEL

Pour piloter deux cent sources sonores indépendantes, il faut du monde : cent amplificateurs stéréophoniques pilotés chacun par l'extension synthétiseur numérique d'un micro-ordinateur Yamaha CX5 (standard MSX). Pour les synchroniser, les cent micro-ordinateurs sont reliés entr'eux par l'interface MIDI intégrée à l'extension synthé. L'ensemble du réseau est supervisé par un COMPAQ (compatible IBM-PC) muni d'une interface MIDI. Pour réduire l'encombrement de l'ensemble (et pour décourager les éventuels vandales), ne seront conservées des micro-ordinateurs que les parties essentielles, montées en racks étanches pour des raisons de fiabilité et de durée de vie.

Pour laisser à l'artiste toute liberté de se déplacer dans l'espace sonore pendant sa phase de création/mise au point, un poste de contrôle sans fil, composé d'un émetteur radio et d'un mini récepteur télé-couleur, a été préféré au cordon ombilical et au clavier d'un terminal classique ; coté micro "superviseur", le récepteur radio est relié à une carte PIA, et la sortie moniteur couleur est branchée sur un petit émetteur VHF. L'émetteur radio (une télécommande 8 canaux pour modèle réduit) est muni de huit interrupteurs trois pôles (seize "touches de fonction") et de deux "joysticks" qui présentent l'avantage de pouvoir faire des réglages "à vitesse variable" et de piloter "à la souris" un curseur à l'écran.

LE LOGICIEL

C'est le FORTH qui a été choisi comme langage de développement pour l'ensemble de l'application. Je ne m'étendrai pas ici sur ses avantages connus: compacité, rapidité, extensibilité, modularité, portabilité et facilité à déboguer. Ce qui est un peu moins connu, c'est sa capacité à s'auto-régénérer complètement, grâce à un "méta-générateur" (ou "cross-compilateur", pour reprendre le terme anglo-saxon).

META-GENERATION

Le concept de méta-génération est en principe assez simple : tout bon "FORTHeur" a déjà utilisé l'assembleur en ligne livré avec son FORTH pour optimiser en langage machine ses routines critiques ; il suffit de l'utiliser pour recréer les "primitives" en langage machine et d'utiliser le Forth lui-même →

pour créer les entêtes du dictionnaire et les "secondaires" exprimées en Forth pour créer un nouveau noyau Forth. Bien sûr, si ce nouveau noyau doit être indépendant de celui qui l'a créé, et surtout s'il doit "tourner" à un emplacement mémoire différent de celui où il a été créé, les outils habituels de construction des définitions doivent être réécris pour tricher un peu avec les adresses mémoire et pour permettre de rendre ce noyau chargeable et exécutable à partir d'une cassette ou d'une disquette, ou même d'être résident en ROM. En fait, la conception d'un méta-générateur n'est pas triviale et demande des connaissances dépassant le niveau moyen des bons "FORTHeurs".

Il est facile maintenant d'imaginer que le méta-générateur puisse être utilisé sur une machine munie d'un processeur de type A, mais avec un assembleur générant du code pour un processeur de type B. C'est de cette manière que, disposant d'un Forth, de son méta-générateur et de plusieurs assembleurs, on peut facilement "porter" Forth sur d'autres types de machines : c'est là que le terme "portabilité" prend toute sa mesure.
Et c'est bien entendu cette caractéristique qui a fait décider du choix du Forth comme langage de développement pour notre application. Disposant d'un méta-générateur de FigForth, le premier travail a donc consisté à porter Forth sur un micro-ordinateur au standard MSX.

Petite parenthèse pour les protectionnistes que j'entends déjà se demander entre les lignes pourquoi n'avoir pas choisi un micro bien français. La réponse est simple : le CX5 est le seul micro (dans une gamme de prix permettant d'envisager un réseau de cent unités) présentant de telles capacités de synthèse sonore: mine de rien, la puce du synthé ne propose pas moins de 8 groupes de 4 oscillateurs, chacun paramètrable indépendamment, tant pour la fréquence que pour l'enveloppe (pour les connaisseurs, un DX7, qui fonctionne sur le même principe de modulation de fréquence, ne dispose que d'un groupe de 6 oscillateurs, permettant bien sûr des timbres plus riches, mais un seul à la fois, alors que le CX5 en permet 8 simultannés).
Malgré cela, oubliez vos sombres pensées, car l'auteur ne tardera pas à s'occuper du cas des autres micros, français compris.

Voici donc le Forth installé sur disque pour le COMPAQ et sur ROM pour les CX5 (autre avantage des MSX au passage : on peut loger 32K octets sur une seule cartouche enfichable, ce qui est plus que large pour loger même une grosse application Forth en ROM, la RAM nécessaire au Forth étant bien sûr celle présente sur l'unité centrale).
Deux interfaces ont été intégrées aux ROMs des CX5: la première permet l'accès direct aux paramètres du synthétiseur (accompagnée d'une petite bibliothèque de fonctions de modulation "lente" -rampe, triangle, aléatoire...- applicables à tout paramètre); la seconde est une interface réseau.

UN RESEAU DE CENT-UN MICROS

Vous ne vous imaginez pas en train programmer cent micros en vous promenant d'un clavier à l'autre ! Comme une interface MIDI (utilisée pour faire communiquer synthés, boîtes à rythmes, séquenceurs, et autres instruments de musique électroniques) est intégrée à l'extension synthé du CX5, c'est ce moyen de communication qui a été utilisé pour relier les CX5 au COMPAQ. Une interface réseau en anneau (simple, mais rapide, servie par interruption mode 2 du Z80) a donc été ajoutée aux noyaux Forth. A partir de là, rien ne s'opposait plus à utiliser les disques du COMPAQ comme mémoire de masse pour tout ce petit monde. Je vous laisse imaginer les facilités de mise au point qui en ont découlé pour les versions suivantes des ROMs.

UN EDITEUR GRAPHIQUE COULEUR A TELECOMMANDE RADIO

La transmission vidéo sans fil n'a pas posé d'autre problème que le bon choix d'un émetteur VHF. L'interface radio ne s'est heurtée qu'à quelques problèmes de parasitage et d'instabilité, vite résolus par une petite fonction de filtrage (un petit problème matériel subsiste encore au niveau de la portée de l'antenne en émission sous tunnel : merci pour les conseils si vous connaissez le problème).
Pour des facilités de développement (et par pitié pour les accus de la radio), une interface clavier permet de simuler l'interface radio.

A partir de là, le plus gros du développement a consisté (et consiste encore) à créer la véritable interface "homme-machine", qui permette d'accéder, en temps réel, à environ quatre cent para-

mètres par synthétiseur, soit quarante mille paramètres ! Et ce à partir de deux joysticks (quatre commandes proportionnelles) et huit interrupteurs (seize commandes tout-ou-rien). Là encore, le Forth s'est montré bien adapté pour créer une base d'objets graphiques. Chaque paramètre est représenté à l'écran par une valeur numérique et un bargraph, et peut être affecté à un joystick, simplement en positionnant le curseur graphique sur la valeur numérique et en actionnant un interrupteur. Le Forth se charge du reste, modifiant la valeur et la représentation bargraph du paramètre à l'écran (et dans un "journal" que l'artiste peut enregistrer chaque fois qu'il désire conserver une trace permanente d'un travail particulièrement réussi, et dont le meilleur sera utilisé pour générer les ROMs définitives), et bien sûr, à travers le réseau, le paramètre correspondant dans le(s) micro(s) à l'écoute sur le réseau, ce paramètre lui-même, à travers différentes "boîtes à engrenages", mettant en route tout un processus visant à changer un ou plusieurs paramètres du synthétiseur, soit statiquement, soit en suivant une fonction de modulation.

UN ENVIRONNEMENT MULTITACHE

C'est à ce niveau qu'intervient une extension multitâche du Forth : un séquenceur, cadencé par un temporisateur du synthé, contôlera sur interruption huit tâches (soit un total de dix tâches par micro en comptant la tâche réseau et une tâche de fond gérant le clavier et l'écran pour contrôle local du micro). Chaque source sonore disposera donc de quatre groupes d'oscillateurs gérés par quatres tâches.

Après mise au point de l'ensemble de l'installation, il va de soi que les outils de développement (radio, editeur, journal) ne seront d'aucune utilité pour son fonctionnement permanent. Quant au COMPAQ, son rôle se limitera à fournir une horloge de référence pour synchroniser l'ensemble et, en cas de problème, à appeler (par modem, bien sûr) les agents du service de maintenance de la RATP, auxquels il fournira un pré-diagnostic de panne afin de limiter autant que possible les temps d'intervention sur site.

FORTH EN PUISSANCE

C'est grâce à la puissance du concept Forth qu'une telle application pourra voir le jour avec des moyens matériels aussi modestes: le CX5 n'est jamais qu'un micro "familial", et l'investissement logiciel pourra s'estimer à environ un homme-an.

L'ensemble de l'installation représentera un potentiel jamais encore atteint dans ce type d'application : deux cent sources sonores servies par mille tâches tournant sur cent processeurs ! L'ambition de Max Neuhaus n'est pas moindre pour le plaisir de vos oreilles...

Voir figure page 20

L'ARTISTE : **Max NEUHAUS** est né au Texas en 1939. Batteur solo de renommée internationale, adepte de John Cage, il enregistre un disque en 1968. Ses réflexions sur les relations artiste-auditeurs l'amènent alors à abandonner concerts et batterie pour créer des "scluptures sonores", pour la plupart dans des lieux de passage grand public (dont trois sont permanentes: Times Square [NewYork 1977], Museum of Contemporary Art [Chicago 1979], Villa Celle [Santomato di Pistoia 1983 Italie], et bientôt: Montparnasse Bienvenüe [Paris 1986], Kergehennec [Bretagne 1986]...).
Max Neuhaus est président de l'ASSOCIATION ECOUTE (12 bis, rue Jules Breton, 75013 PARIS, 42.78.03.84.) qu'il a créée spécialement pour ses projets français. *If you know Forth well enough and wish to join that project, please call or write.*

L'AUTEUR : **Christophe LAVARENNE** est né à Nice en 1956. Ingénieur Arts et Métiers, il conçoit des robots industriels pendant quatre ans. Aujourd'hui ingénieur conseil, il a traduit en français un ouvrage de Tony Hasemer sur Lisp pour micro-ordinateurs et développe des applications en Forth, dont celle présentée dans cet article.

AUTRES REFERENCES :

ASSOCIATION JEDI
8 rue Poirier de Narçay 75014 Paris.

MICROPROCESSOR ENG LTD
21 Hanley Road Shirley, Southampton SO15AP ENGLAND (méta-générateur FigForth pour la plupart des processeurs sur le marché des micro-ordinateurs).

Zohra Hassaïne

POSITIONNEMENT DU PROBLEME

Le programme (écrit en VLISP) présenté ici permet de réaliser les tâches suivantes:

Etant donnée une formule de calcul des propositions

1) Dire si c'est une tautologie, une antilogie ou une formule contingente en utilisant la méthode des polynômes à coefficients dans Z/2Z.

2) Dans le cas où la formule est contingente, donner sa forme normale conjonctive canonique.

3) Donner une forme normale conjonctive réduite; imprimer les clauses correspondantes.

UN PEU DE VOCABULAIRE

Dans l'article qui suit (et même dans l'introduction ci-dessus), il est fait usage d'un certain nombre de termes appartenant à la logique. Ce petit lexique est destiné à ceux qui sont peu familiers de ce domaine.

- Sujet : premier terme d'une proposition logique.

- Prédicat : second terme d'une proposition logique.

- Tautologie : proposition logique qui reste vraie quelque soit la valeur de vérité des propositions qui la composent.

- Antilogie : proposition logique qui reste fausse quelque soit la valeur de vérité des propositions qui la composent.

- Contingente : se dit d'une proposition logique qui peut être vraie ou fausse suivant la valeur de vérité des propositions qui la composent.

- forme normale conjonctive canonique : combinaison de structure définie entre les variables logiques d'une proposition ou d'une fonction qui rend vrai la proposition ou la fonction. La structure est composée de termes liés par des <u>ou</u> logiques, chacun de ces termes est constitué des <u>n</u> variables ou de leur complément liés par des <u>et</u> logiques.

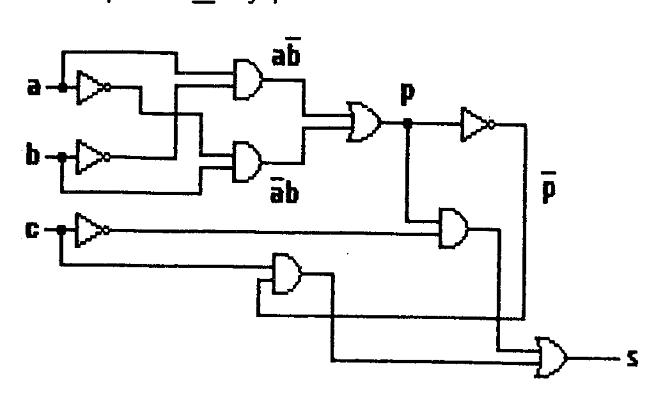


figure 0 .

Exemple : la fonction de la figure 0 est la partie sommation d'un additionneur 1 bit dont la table de vérité est représentée ci dessous avec sa forme normale conjonctive canonique:

a	b	c	s	
0	0	0	0	
0	0	1	1	-> $\bar{a}.\bar{b}.c$
0	1	0	1	-> $\bar{a}.b.\bar{c}$
0	1	1	0	
1	0	0	1	-> $a.\bar{b}.\bar{c}$
1	0	1	0	
1	1	0	0	
1	1	1	1	-> $a.b.c$

On a donc la fncc

$s = \bar{a}.\bar{b}.c + \bar{a}.b.\bar{c} + a.\bar{b}.\bar{c} + a.b.c$

NOTICE D'UTILISATION

- On peut rentrer une formule ou une liste de formules qui seront placées dans un fichier.

- Les formules doivent être rentrées sous forme infixe entièrement parenthèsée.

- Pour les connecteurs, il faut utiliser:

```
ou   pour  V
et   pour  Λ
imp  pour  -->
equ  pour  <-->
non  pour  ¬
```

- Le programme est lancé par la fonction <u>pro</u> qui appelle la fonction <u>fichier</u> si les formules sont dans un fichier et qui appelle aussi la fonction <u>projet</u> qui lance réellement le programme en répondant aux trois questions du projet.

- Donc pour lancer le programme, il faut taper: (pro)

EXPLICATION DU PROGRAMME

- On donne une formule de calcul des propositions sous forme infixe entièrement paranthèsée. Elle sera manipulée comme une liste implémentée par un arbre binaire.

Exemple:

Soit la formule: (((p equ q)imp r)imp r imp p)
pour (((p <--> q)--> r)--> r)--> p)
En mémoire cette liste est représentée par l'arborescence de la figure 1

- Le programme cacule la forme polonaise de la formule ainsi rentrée grâce à la <u>fonction polo:</u>

<u>méthode utilisée:</u>

. Soit la formule rentrée sous forme de liste. Le <u>car</u> de la liste est la partie gauche de la formule, le <u>cadr</u> de la liste est le connecteur, le <u>cddr</u> est la partie droite.

. Si le <u>cddr</u> existe le connecteur est binaire, donc on place dans une nouvelle liste le connecteur (cadr), la partie gauche (car) et la partie droite (cddr)

. Sinon le connecteur est unaire (non), donc on place le connecteur puis la partie droite dans une nouvelle liste.

Exemple: La forme polonaise d'une formule:

polo(((p equ q)imp r)imp r)imp p)
= (imp(imp(imp(equ p q)r)r)p)

- Nous gardons cette forme polonaise parenthèsée parce que nous la manipulerons ainsi pour donner son polynôme à coefficients dans Z/2Z, pour simplifier et pour donner la forme normale conjonctive canonique et la forme réduite de la formule.

- La forme polonaise est une liste d'atomes et de sous-listes.

- A partir de cette forme polonaise, nous calculons le polynôme de la formule à coefficients dans Z/2Z grâce à <u>la fonction poly:</u>

<u>méthode utilisée:</u>

. Soit la liste de la forme polonaise de la formule.

. Dans le polynôme associé à la formule le signe "+" est remplacé par un blanc et le signe "x" est remplacé par une double parenthèse:

Exemple: 1 + X + X x Y
= (1 X ((X)Y))

. On teste le <u>car</u> de la liste de la forme polonaise de la formule. Selon que c'est un connecteur ou une variable on construit <u>poly</u> par récurrence sur les formules:

```
poly (X)        = (X)
poly (¬ A)      = (1 poly (A))
poly (Λ A B)    = ((poly (A)) poly (B))
poly (V A B)
      = (poly (A) poly (B) ((poly (A)) poly (B)))
poly (--> A B)
           = (1 poly (A) ((poly (A)) poly (B)))
poly (<--> A B) = (1 poly (A) poly (B))
```

où X est une variable de type atome et A et B des sous-listes représentant des sous-formules de forme polonaise.

Exemple:

poly (et (non a) (ou b c))
= ((poly (non a)) poly (ou b c))
= ((1 poly a) poly b poly c ((poly b) poly c))
= ((1 a) b c ((b) c))

- Pour simplifier le polynôme, on utilise la <u>fonction simplification</u>. Plus exactement la fonction <u>simplification</u> prend en paramètre la forme polonaise parenthèsée de la formule, appelle <u>poly</u> qui calcule le polynôme (comme nous l'avons vu précédemment).

schéma de la fonction <u>simplification</u>:

```
(simplification f) (=) (simpladd (simplif f))
(simplif f) (=)
    (enleveliste (recherchemult (simpl (poly f))))
```

Explication de chacune des fonctions appelées:

- La fonction <u>simpl</u>: Elle repère chaque sous-liste (multiplication du polynôme et les développe en utilisant la distributivité. Elle fait appel à la fonction <u>dist</u> qui fait appel à la fonction <u>developpe</u> qui fait appel à la fonction <u>mult</u>.

Exemple:

(simpl (1 X ((X 1) Y X)))
retourne (1 X (X 1 Y) (X 1 X))

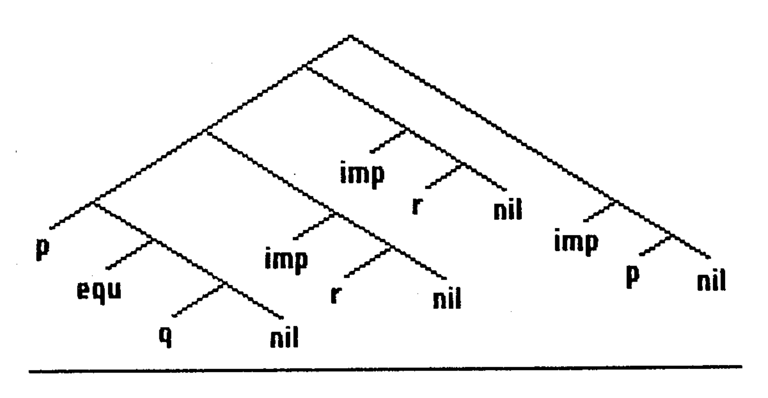


figure 1.

ce qui est équivalent en logique à :

1 + X + (X x 1 x Y) + (X x 1 x X)

- La fonction recherchemult: Elle recherche les multiplications dans la liste rendue par simpl, multiplications qui sont des sous-listes de la liste. Une fois que l'on a repéré une sous-liste, on appelle simplmult, fonction qui simplifie les multiplications en utilisant les propriétés de la multiplication dans Z/2Z, c'est à dire:

X x 1 = X , X x X = X et X x 0 = 0

Exemple:

(recherchemult (1 X (X 1 Y) (X 1 X)))
retourne (1 X (X Y) (X))

- La fonction enleveliste: Elle supprime les sous-listes formées d'un atome (variable) rendues après la simplification des multiplications.

Exemple:

(enleveliste (1 X (X Y) (X)))
retourne (1 X (X Y) X)

- La fonction simpladd: Elle fait la simplification finale c'est à dire celle de l'addition. Elle fait appel à la fonction simpad qui fait appel à la fonction compare qui fait appel à la fonction comp. La simplification se fait en utilisant les propriétés de l'addition dans Z/2Z c'est à dire: X + X = 0 (0 élément neutre pour +). Les atomes sont comparés ensemble et les sous-listes de même longueur sont comparées ensemble.

Exemple:

(simpladd (1 X (X Y) Z X (Y X) Y))
retourne (Y Z 1)

En résumé la fonction simplification appliquée à une formule polonaise parenthèsée retourne le polynôme associé à cette formule à coefficients dans Z/2Z entièrement simplifié. Ainsi:

(1) s'il est égal à 1, la formule est une tautologie.
(2) s'il est égal à 0 (nil), la formule est une antilogie.
(3) s'il est égal à autre chose, la formule est contingente.

Ces tests sont faits dans la fonction projet (en début de programme) et dans le cas (3) on appelle les fonctions fncc (pour calculer la forme normale conjonctive canonique), affiche (pour la forme réduite) et clause (pour imprimer les clauses).

Méthode utilisée dans fncc:

. C'est par la table de vérité d' une formule que l'on construit sa forme normale conjonctive canonique.

.Tout d'abord on recherche les variables de la formule polonaise grâce aux fonctions aplatie et rechvar appelées dans la fonction vara.
aplatie prend en argument une liste (formule polonaise) avec des sous-listes et retourne la liste des atomes de la liste.
rechvar retourne les variables contenues dans la liste donnée par aplatie dans l'ordre.

. Puis grâce à la fonction table, on construit la liste des différentes combinaisons possibles de valeurs de vérité des variables.
table: Les deux valeurs possibles pour une variable sont 1 et 0 (pour vrai et faux) donc au départ table prend pour argument la liste (1 0) et n qui est le nombre de variables de la formule. Ce nombre est donné par: (length (vara l)) (l étant la formule polonaise proposée).

table retournera les 2^n combinaisons possibles.

Au niveau de la programmation l'idée de table est la suivante: à partir de la liste (1 0) on forme des sous-listes en rajoutant 1 et 0 à chaque élément. Quand le nombre d'éléments des sous-listes atteint n on arrête.

Exemple: (table (1 0) 3)

3 représente le nombre de variables.
table doit renvoyer 2^3 = 8 combinaisons.
La façon dont elle procède est représentée figure 2, lorsque la longueur de chaque sous-liste est 3 on arrête et on obtient 8 combinaisons

. Ensuite, grâce à la fonction distribution, on attribue les valeurs de vérité de chaque combinaison (sous-liste) de table aux variables de la formule.
distribution est appelée avec trois paramètres
- l : la formule
- var : la liste des variables de l (vara l) (pour pouvoir les reconnaitre dans la formule)
- dist : une combinaison de valeurs de vérité.

Exemple:

(distribution (X imp Y) (X Y) (1 0))
retourne (1 imp 0)

tabverite teste la valeur (0 ou 1) que prend la formule pour les différentes combinaisons et grâce à la fonction fornum, on obtient la liste des combinaisons de table qui rendent la formule fausse (=0).
Il ne reste plus qu'à affecter les variables aux éléments (0 ou 1) des combinaisons et rajouter les connecteurs grâce aux fonctions forme et dijon et l'on obtiendra la forme normale conjonctive canonique. (pour 0 on rend p et pour 1 on rend np (= non p)).
forme: Cette fonction accepte 2 arguments:
- l1 : la liste des variables de la formule.
- l2 : la liste des combinaisons (évaluations) qui rendent fausse la formule.
Elle retourne la forme normale conjonctive canonique en utilisant le fonction dijon

ETUDE DES FONCTIONS

Cette étude explique les entrées et les sorties des fonctions.
(e est utilisé pour entrée et s pour sortie)

Programme principal

- Les fonctions pro et projet font tourner le programme, fichier est utilisée au cas où les données seraient dans un fichier.

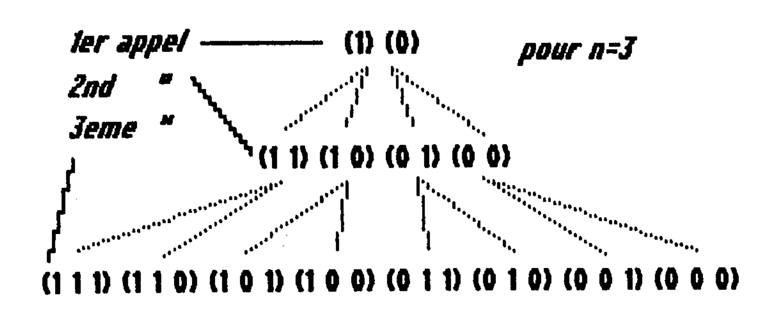


figure 2.

pour chacune des sous-listes on rajoute 0 d'une part et 1 d'autre part pour former deux nouvelles listes. On itere ce procede n fois. (n est le nombre de variable)

pro :

e : sans paramètre
s : (lance tous le programme)

fichier :

e : une liste de plusieurs formules
s : pour chaque formule, teste si tautologie...

projet :

e : une liste (= formule infixe)
s : son polynôme, sa forme normale conjonctive canonique, sa forme normale conjonctive réduite, les clauses.

FORME POLONAISE

polo :

e : une liste (= formule infixe)
s : sa forme polonaise (parenthèsée)

TAUTOLOGIE

poly :

e : une liste (= formule polonaise parenthèsée)
s : son polynôme dans Z/2Z

simpl :

e : une liste (= polynôme dans Z/2Z d'une formule)
s : polynôme simplifié

list :

e : 2 listes (la 1ere est en facteur avec la 2nd)
s : distribution de la 1ere liste par rapport à la 2nd

devellope :

e : 2 listes ou 1 atome et 1 liste (X Y) (A B)
s : une liste ((X Y A) (X Y B))

mult :

e : 2 listes ou 1 liste et un atome ou 2 atomes
X (A B)
s : 1 liste ((X A) (X B))

simplmult :

e : 1 liste (A A B 1 C)
s : 1 liste (A B C)

simpmu :

e : 1 atome et 1 liste A (A B 1 C)
s : 1 liste (A B 1 C) (fait la simplification de la multiplication

recherchemult :

e : 1 liste (le polynôme de la formule)
s : 1 liste (repère les sous-listes et les simplifie)

simpladd :

e : 1 liste ((A B) C (D E) C A (D E))
s : cette liste est simplifiée grâce aux propriétés de l'addition dans Z/2Z (A (A B))

simpad :

e : 2 listes ou 1 atome et 1 liste (le polynôme)
s : la 2eme liste privée ou non de l'élément de 1er argument

comp :

e : 2 listes
s : vrai ou faux (t ou nil) selon que les deux listes sont égales ou non

compare :

e : 2 listes ou 2 atomes ou 1 liste et 1 atome
s : teste si les deux arguments sont égaux, retourne t ou nil

enlevelist :

e : 1 liste (le polyrôme simplifié)
s : cette liste en remplaçant les sous-listes d'un atome par l'atome lui-même

simplif :

e : 1 liste (la forme polonaise parenthèsée)
s : le polynôme de la formule dont les multiplications sont simplifiées

simplification :

e : 1 liste (la forme polonaise parenthèsée)
s : le polynôme entièrement simplifié

FORME NORMALE CONJONCTIVE CANONIQUE

aplatie :

e : 1 liste (= formule)
s : la liste d'atomes qui composent la liste entrée (sans sous-liste)

rechvar :

e : 1 liste (= 1 formule sans sous-formules)
s : la liste des variables (dans l'ordre) de la formule

table :

e : la liste (1 0) et 1 nombre n (= nombre de variables)
s : liste de 2^n sous listes (= combinaison des valuations des variables)

distribution :

e : une formule polonaise, 1 liste de variable et 1 combinaison de valuation
s : la formule avec à la place des variable leur valuation

tabverite :

e : une formule polonaise avec les valeurs de vérité à la place des variables
s : 0 ou 1 suivant que la formule est fausse ou vraie

vara :

e : une formule
s : la liste de ses variables avec leur ordre d'apparition dans la formule

fornum :

e : 1 formule, ses n variables (liste) et les 2^n combinaisons
s : la liste des comninaisons qui rendent la formule fausse

fncc :

e : une formule polonaise (parenthèsée)
s : sa forme normale conjonctive canonique

forme :

e : la liste des variables de la formule et la liste des combinaisons qui rendent la formule fausse
s : la forme normale conjonctive canonique avec variables et connecteurs

dijon :

e : la liste des variables et une sous-liste de la liste des combinaisons de valuation
s : 1 liste : disjonction des variables

FORME REDUITE

reduc :

e : 1 liste qui est la fncc d'une formule sans les connecteurs
s : la forme reduite de la fncc

reduc1 :

e : deux listes (au 1er appel = 2 fois la fncc)
s : la liste des réductions de la fncc (forme réduite de la fncc)

reduc2 :

e : 1 sous-liste de la fncc et la fncc (au 1er appel)
s : la liste des réductions de l'élément avec tous les autres

controle :

e : 2 listes (1 liste de réduction et 1 liste des éléments de la fncc qui ont donné une réduction)
s : la liste des réductions

compact :

e : 1 liste (la fncc)
s : la liste sans éléments en double

res :

e : 2 éléments (listes) à réduire
s : la liste vide si les éléments sont égaux, leur réduction sinon

comparaison :

e : 2 éléments à réduire
s : 1 liste avec "oui" "non" suivant que des variables sont présentent ou non dans les deux listes

verif :

e : la liste ("oui" "non") et n le nombre qui compte les "non"
s : n le nombre de "non"

resulreduc :

e : deux éléments à réduire
s : la réduction

résultat :

e : 1 liste (élément de la fncc) et la liste "oui" "non"
s : 1 liste (= réduction sans la variable qui a donné "non"

enlev1, **enlev2**, **affiche**, **aff**, **supprime** et **supp** arrangent les formules pour bien les présenter aux fonctions (sans les connecteurs) et aux utilisateurs (en supprimant les nil,en remettant les connecteurs pour la compréhensions des formules)

CLAUSES

clause :

e : 1 liste de réduction : ((p np) (p q))
s : 1 liste de clauses : ((q => p) (=> p q))

cl1 et **cl2** :

permettent la formation de ces clauses

- Après l'obtention de la forme normale conjonctive de la formule, le procédé de réduction est le suivant:

- On considère la fncc de la formule sans les connecteurs Λ et V que l'on envoie à la fonction reduc. reduc appelle la fonction reduc1 avec au premier appel les deux arguments de reduc1 qui sont deux fois la fncc.

- chaque élément sera comparé à tous les autres et l'on recueillera toutes les réductions qu'il aura pu donner avec d'autres éléments différents dans une liste.

- On utilise deux paramètres dans reduc1 pour que tous les éléments de la fncc puissent être comparé avec tous les autres.

- reduc1 appelle ensuite reduc2 qui appelle res etc... (voir le listing). La comparaison de deux éléments est faite selon la tautologie:
((p V q) ^ (p V q)) (--> p

- On prend deux éléments (2 sous-listes), si un élément est présent dans les deux listes, on met "oui" sinon on met "non". S'il y a un "non" on peut appliquer la tautologie sinon il n'y a pas de réduction possible. Ces opérations sont faites grâce aux fonctions comparaison, verif, resulreduc et resultat.

- Les fonctions enlev1, enlev2, affiche, aff, supprime, supp servent à enlever ou à remettre des connecteurs pour présenter les formules de façon correcte par rapport aux autres fonctions.

- Enfin, à partir de la forme réduite, on fabrique les clauses correspondant aux listes.

LE PROGRAMME:

```
(de pro ()
(tyoesc 69)
(terpri)
(print "VOULEZ VOUS RENTRER LES DONNEES PAR FICHIER (o)ui/(n)on ")
(if (eq (read) 'n)
(progn
(terpri)
(print "                    DONNEZ VOTRE EXPRESSION EN INFIXEE")
(terpri)
(let ((ff (read)))
(print ff)
(projet ff) (terpri)
(print "VOULEZ-VOUS CONTINUER  o/n ?") (terpri)
(if (eq (read) 'n) (print "                    FIN") (pro)))
(progn
(input 'f2)
(let ((ff (read)))
(fichier ff)))))

(de fichier (l)
(if (null l)(print "FIN DU FICHIER")
(progn
(terpri)
(print"VOTRE FORMULE")
(print (car l))
(terpri)
(projet (car l))
(print"*********************************************")
(fichier (cdr l)))))
```

```
(de projet (l)
(terpri)
(let ((ff (polo l)))
     (let ((tauto (simplification ff)))
          (cond ((equal tauto '(1))
                       (print "          C'EST UNE TAUTOLOGIE"))
                ((equal tauto nil)
                       (print "          C'EST UNE ANTILOGIE"))
                (t     (print "          FORMULE CONTINGENTE, VOICI SON POLYNOME")
                    (terpri) (print tauto)
                    (terpri) (let (( fconj (fncc ff)))
                     (print "          FORME NORMALE CONJONCTIVE CANONIQUE")
                    (terpri)
                     (print fconj) (terpri)
                     (let ((reduction (affiche (enlev1 (reduc (supprime fconj))))))
                     (print "          FORME NORMALE REDUITE")
                     (terpri)
                     (print reduction)(terpri)
                     (print "          VOICI LES CLAUSES")(terpri)
                     (print (clause (enlev1 (reduc (supprime fconj)))))))))))





(de polo (x)
(cond ( (null x) nil)
      ( (atom x) x)
      ((gau x) [(op x) (polo(gau x)) (polo(droi x)) ])
      (t [(op x) (polo(droi x))])
)))

(de op(x)
     ( if (cddr x)(cadr x)(car x))
)

(de gau (x)
     (if (cddr x) (car x) nil)
) (de droi (x) (if (cddr x)(caddr x)(cadr x)))

(de poly (l)
(cond ((null l) ())
      ((listp l) (selectq (car l)
         ('ou (append (append (poly (cadr l)) (poly (caddr l)))
                  (cons (append [(poly (cadr l))] (poly (caddr l))) nil)))
         ('et (cons (append [(poly (cadr l))] (poly (caddr l))) nil))
         ('imp (append (append 1 (poly (cadr l)))
                  (cons (append [(poly (cadr l))] (poly (caddr l))) nil)))
         ('equ (append (append 1 (poly (cadr l)))
                  (poly (caddr l))))
         ('non (append 1 (poly (cadr l))))
         (t [l])))
      (t [l]))))

(de simpl (l)
  (if (null l) ()
  (let ((j (car l)))
      (append  (if (atom j) [j]
               (dist (simpl(car j)) (simpl(cdr j))))
               (simpl(cdr l))))))

(de dist (l1 l2)
    (if (null l1) ()
        (append (developpe (car l1) l2) (dist (cdr l1) l2))))

(de developpe (x l)
   (if (null l) ()
        (cons (mult x (car l)) (developpe x (cdr l)))))

(de mult (x1 x2)
    (if (atom x1) (if (atom x2) [x1 x2]
                    (cons x1 x2))
        (atom x2) (cons x2 x1)
                  (append x1 x2)))


(de simplmult (l)
     (if (null (cdr l)) l
              (if (equal (car l) '1) (simplmult (cdr l))
                  (simpmu (car l) (simplmult (cdr l))))))

```

```
(de simpmu (x l)
    (cond ((null l) [x])
          ((equal x (car l)) (simpmu x (cdr l)))
          ((equal (car l) 1) (simpmu x (cdr l)))
          (t (append [(car l)] (simpmu x (cdr l))))))

(de recherchemult (l)
     (if (null l) ()
              (append (if (listp (car l)) [(simplmult (car l))] [(car l)])
                      (recherchemult (cdr l))))))

(de simpladd (l)
     (if (null (cdr l)) l
         (let ((j (car l))) (simpad j (simpladd (cdr l))))))


(de simpad (x l)
      (cond ((null l) [x])
            ((compare x (car l)) (cdr l))
            (t (append [(car l)] (simpad x (cdr l))))))

(de comp (x1 x2)
      (cond ((null x1) t)
            ((member (car x1) x2) (comp (cdr x1) x2))
            (t ()))))


(de compare (x1 x2)
      (if (atom x1) (cond ((listp x2) nil)
                          ((equal x1 x2) t)
                          (t nil))
                    (cond ((atom x2) nil)
                          ((equal (length x1) (length x2)) (comp x1 x2))
                          (t nil))))


(de enleveliste (l)
     (if (null l) ()
              (append (if (and (listp (car l))
                               (equal (length (car l)) 1)) (car l) [(car l)])
                      (enleveliste (cdr l)))))



(de simplif (l)
   (enleveliste (recherchemult (simpl (poly l)))))


(de simplification (l)
    (simpladd (simplif l)))


(de aplatie (l)
    (if (null l) ()
                (let ((j (car l)))
                     (append (if (atom j) [j]  (aplatie j))
                                         (aplatie (cdr l))))))


(de rechvar (l1 l2)
    (if (null l1) l2
        (let ((j (car l1)))
         (if (or (equal j 'ou) (equal j 'et) (equal j 'imp)
                 (equal j 'non) (equal j 'equ) (member j l2))
             (rechvar (cdr l1) l2)
             (rechvar (cdr l1) (append l2 [j]))))))


(de table (l n)
     (if (null l) ()
(if (equal n '1) l
         (let ((j (car l)))
   (if (equal (length j) n) l (append(table(cons(append(if (atom j) [j] j) [1])
                                                  [(append(if (atom j) [j] j) [0])])
                                            n)
                                  (table (cdr l) n)))))))


```

```
(de distribution (l var dist)
 (if (null l) ()
      (let ((j (car l)))
 (if (atom j) (cond((or (equal j 'ou) (equal j 'et)(equal j 'imp)
                        (equal j 'equ) (equal j 'non))
                        (append [j] (distribution (cdr l) var dist)))
                       ((equal j (car var)) (append
                 (if (listp dist)[(car dist)][ dist])
                                    (distribution (cdr l) var dist)))
                      (t (distribution l (cdr var) (cdr dist))))
       (append [(distribution j var dist)] (distribution (cdr l) var dist))))))


(de tabverite (l)
 (cond ((null l) ())
        ((listp l) (selectq (car l)
         ('ou (if (and (equal (tabverite (cadr l)) '0)
                        (equal (tabverite (caddr l)) '0)) '0 '1))
         ('imp (if (and (equal (tabverite (cadr l)) '1)
                         (equal (tabverite (caddr l)) '0)) '0 '1))
         ('et (if (and (equal (tabverite (cadr l)) '1)
                        (equal (tabverite (caddr l)) '1)) '1 '0))
         ('non (if (equal (tabverite (cadr l)) '1) '0 '1))
         ('equ (if (equal (tabverite (cadr l)) '1)
                   (if (equal (tabverite (caddr l)) '1) '1 '0)
                   (if (equal (tabverite (caddr l)) '0) '1 '0)))
         (t l)))
        (t l)))


(de vara (l)
(rechvar (aplatie l) nil))

(de fornum (l1 var dist)
(if (null dist) ()
(if (equal (tabverite (distribution l1 var (car dist))) '0)
(append [(car dist)](fornum l1 var (cdr dist)))
(fornum l1 var (cdr dist)))))

(de fncc (l)
(forme (vara l) (fornum l (vara l) (table  '(1 0) (length  (vara l))))))

(de forme (l1 l2)
(if (null l2) ()
(ifn (null (cdr l2))
(append [(dijon l1 (car l2)) '/\ ](forme l1 (cdr l2)))
(append [(dijon l1 (car l2)) ](forme l1 (cdr l2))))))

(de dijon (x1 x2)
(if (null x2) ()
 (if (atom x2)
                 (if (equal x2 '0) (car x1) (implode [ 'non (car x1)]))
(ifn (null (cdr x2))
(if (equal (car x2) '0)
(append [(car x1) '\/ ](dijon (cdr x1) (cdr x2)))
(append [ (implode ['non (car x1)]) '\/ ](dijon (cdr x1) (cdr x2))))
(if (equal (car x2) '0)
(append [(car x1)] (dijon (cdr x1) (cdr x2)))
(append [(implode [ 'non (car x1)])](dijon (cdr x1) (cdr x2))))))))



(de reduc (l)
    (compact l)
    (let ((l1 (reduc1 l l)))
         (ifn (equal l1 l) (reduc l1 ) (compact l1))))

(de reduc1  (l1 l2)
    (if (null l1) () (append(reduc2 (car l1) l2) (reduc1 (cdr l1) l2))))

(de reduc2 (x l1 l2 l3)
    (if (null l1)  (let (( cont (controle l2 l3))) (if (null cont) [x] cont))
        (let ((r (res x (car l1))) (suite (cdr l1)))
        (cond ((or (member r l2) (equal r nil)) (reduc2 x suite l2 l3))
              ((equal r x) (if (member x l3) (reduc2 x suite l2 l3)
                                      (reduc2 x suite (append l2 [r]) l3)))
              (t (reduc2 x suite (append l2 [r]) (append l3 [x])))))))

```

```
(de controle (l1 l2)
    (if (null l1) ()
        (let ((j (car l1)))
        (append (if (member j l2) () [j]) (controle (cdr l1) l2)))))
(de compact (l)
 (if (null l) ()
  (let ((j (car l)) (h (cdr l)))
      (append (if (member j h) () [j]) (compact h)))))
(de res (l1 l2)
(cond ((and (atom l1) (listp l2) (member l1 l2)) l1)
      ((and (atom l2) (listp l1) (member l2 l1)) l2)
      ((and (atom l1) (atom l2) (equal l1 l2)) ())
      ((and (listp l1) (listp l2) (equal l1 l2)) ())
((and (listp l1) (listp l2) (equal (length l1) (length l2)))(resultreduc l1 l2))(t ()
(de comparaison (l1 l2)
  (if (null l1) ()
     (append (if (equal (car l1) (car l2)) ['oui] ['non])
             (comparaison (cdr l1) (cdr l2)))))
(de verif (l n)
    (if (null l) n
        (if (equal (car l) 'non) (verif (cdr l) (+ n 1)) (verif (cdr l) n))))
(de resultreduc (l1 l2)
    (let ((c (comparaison l1 l2)))
         (if (equal (verif c 0) '1) (resultat c l1) l1)))
(de resultat (l1 l2)
    (if (null l1) ()
      (append (if (equal (car l1) 'oui) [(car l2)] [()])
              (resultat (cdr l1) (cdr l2)))))
(de enlev1 (l)
    (if (null l) ()
        (let ((j (car l))) (append [(enlev2 j)] (enlev1 (cdr l))))))
(de enlev2 (l)
    (if (null l) ()
        (let ((j (car l)))
             (append (if (equal j nil) () [j]) (enlev2 (cdr l))))))
(de affiche (l)
    (if (null l) ()
        (append(append [(aff (car l))] (ifn (null (cdr l)) ['/\] ()))
                (affiche (cdr l)))))
(de aff (l)
   (if (null l) ()
    (let ((j (car l)) (h (cdr l)))
        (append (append  [j] (ifn (null h) ['\/] ())) (aff h)))))
(de supprime (l)
    (if (null l) ()
        (append (if (equal (car l) '/\)  () [(supp(car l))] )
                (supprime (cdr l)))))
(de supp (l)
    (if (null l) ()
      (append (if (equal (car l) '\/) () [(car l)])
              (supp (cdr l)))))
(de clause (l)
          (if (null l) ()
         (append[(cl1 (car l))] (clause (cdr l)))))
(de cl1 (l1 l2 l3)
  (if (null l1) (cl2 l2 l3)
        (let ((exp (explode(car l1))))
             (if (equal (car exp) 'n ) (cl1 (cdr l1) (append l2 (cdr exp) ) l3)
                 (cl1 (cdr l1) l2 (append l3 exp))))))
(de cl2 (l1 l2)
  (append (append l1 ['=>] ) l2))
```

par Michel LANGLOIS

TURBO-PACSAL

Le TURBO-PASCAL CPM80 pour l'AMSTRAD (ou SCHNEIDER) ne comporte pas de routines spécifiques. En voici quelques-unes.

La place disponible pour le programme source étant de huit K environ, il est impossible de tout faire tenir en mémoire, c'est pourquoi j'ai opté pour une séparation en blocs spécialisés et appelés TOOLS.

Il y a cinq TOOLs:

- TOOLS1 — il devra être chargé à chaque utilisation.
- TOOLS2 — ne concerne que les ordres graphiques.
- TOOLS3 — ne concerne que les ordres textes.
- TOOLS4 — est une routine de tracé de cercles. TOOLS2 devra être chargé avant chaque utilisation.
- TOOLS5 — reconstruction des fonctions SIN et COS. TOOLS2 devra être chargé. J'ai utilisé la méthode de BYTE (80) pour le FORTH.

Queques remarques sur certaines procédures:

TOOLS1

CUR_ON et CUR_OFF servent à supprimer le système lors d'une boucle <u>repeat</u>. Le curseur réaparait à la fin.

TOOLS2

TEST donne le numéro du crayon.
Pour créer une fenêtre graphique, il faut fixer l'origine avec 'ORIGIN', simple non!.
CLG efface l'écran graphique avec la couleur spécifiée par GPAPER.

TOOLS3

Pour utiliser les fenêtres texte, la procédure est la suivante:

'WINDOW(1,10,40,10,20)' 'PEN(1)' 'PAPER(0)'

défini une fenêtre numéro 1 avec un crayon numéro 1 et un papier numéro 0. L'écriture se fera dorénavant dans la fenêtre sélectionnée ou dernièrement définie. L'activation d'une fenêtre est réalisée par 'SETWINDOW(n)'. L'effacement de son contenu est réalisée par 'CLRSCR' de TURBO.

Les autres TOOLS se passent de commentaire, car ils sont inclus. N'oubliez pas, pour les utiliser dans vos propres programmes, de les inclure également (pour les débutants en TURBO sur AMSTRAD, inclure signifie prendre en compte le contenu du fichier spécifié; ceci permet de compiler un programme source ne pouvant normalement pas tenir en mémoire vive, car celle-ci est limitée pour l'AMSTRAD à 8k utiles sous TURBO).

J'espère que vous trouverez ces procédures utiles, même si le jeu de fonctions disponibles reste incomplet, car de mon coté il y a encore fort à faire.

```
(***********************************************)
(*  TOOLS1.PAS   Le 12 oct 1985 (c) LANGLOIS MICHEL *)
(***********************************************)

procedure mode(m: integer);        ( MODE(n);                  )
 begin
     inline    ($3A/M/             ( LD A,(M)                  )
                $CD/$BE9B/         ( CALL #BE9B 'FIRMAWARE'    )
                $BC0E)             ( DEFW #BC0E 'MODE'         )
 end;

procedure trans(c: integer);       ( TRANS(n);                 )
 begin
     inline    ($3A/C/             ( LD A,(C)                  )
                $CD/$BE9B/         ( CALL #BE9B                )
                $BB9F)             ( DEFW #BB9F 'TRANS'        )
 end;

procedure ink(i,c1,c2: integer); ( INK(i,n1,n2);               )
 begin
     inline    ($3A/C1/            ( LD A,(C1)                 )
                $47/               ( LD B,A                    )
                $3A/C2/            ( LD A,(C2)                 )
                $4F/               ( LD C,A                    )
                $3A/I/             ( LD A,(I)                  )
                $CD/$BE9B/         ( CALL #BE9B 'FIRMWARE'     )
                $BC32)             ( DEFW #BC32 'INK'          )
 end;

procedure speedink(c1,c2: integer); ( SPEEDINK(t1,t2);         )
 begin
     inline    ($3A/C1/            ( LD A,(C1)                 )
                $67/               ( LD H,A                    )
                $3A/C2/            ( LD A,(C2)                 )
                $6F/               ( LD L,A                    )
                $CD/$BE9B/         ( CALL #BE9B                )
                $BC3E)             ( DEFW #BC3E 'SPEEDINK'     )
 end;

procedure border(c1,c2: integer);( BORDER(n1,n2);              )
 begin
     inline    ($3A/C1/            ( LD A,(C1)                 )
                $47/               ( LD B,A                    )
                $3A/C2/            ( LD A,(C2)                 )
                $4F/               ( LD C,A                    )
                $CD/$BE9B/         ( CALL #BE9B 'FIRMWARE'     )
                $BC38)             ( DEFW #BC38 'BORDER'       )
 end;

procedure flyback;                 ( pas d'argument            )
 begin
     inline    ($CD/$BE9B/         ( CALL #BE9B                )
                $BD19)             ( DEFW #BD19 'FLYBACK'      )
 end;

procedure graphic(c: integer);     ( GRAPHIC(n);(* n=0,1-3*))
 begin
     inline    ($3A/C/             ( LD A,(C)                  )
                $CD/$BE9B/         ( CALL #BE9B                )
                $BC59)             ( DEFW #BC59 'GRAPHIC'      )
 end;

procedure cur_on;
 begin
     inline    ($CD/$BE9B/         ( CALL #BE9B                )
                $BB7B)             ( DEFW #BB7B 'CUR_ON'       )
 end;

procedure cur_off;
 begin
     inline    ($CD/$BE9B/         ( CALL #BE9B                )
                $BB7E)             ( DEFW #BB7E 'CUR_OFF'      )
 end;
```

```
(***************************************************)
(* TOOLS2.PAS  Le 12 oct 1985 (c) LANGLOIS Michel *)
(***************************************************)

procedure gpen(i: integer);       ( GPEN(n);                     )
 begin
     inline    ($3A/I/            ( LD A,(I)                     )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBDE)            ( DEFW #BBDE 'GPEN'            )
 end;

procedure gpaper(i: integer);     ( GPAPER(n);                   )
 begin
     inline    ($3A/I/            ( LD A,(I)                     )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBE4)            ( DEFW #BBE4 'GPAPER'          )
 end;

procedure gmove(x,y: integer);     ( MOVE(x,y);                  )
 begin
     inline    ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBC0)            ( DEFW #BBC0 'MOVE'            )
 end;

procedure gmover(x,y: integer);    ( MOVER(dx,dy);               )
 begin
     inline    ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBC3)            ( DEFW #BBC3 'MOVER'           )
 end;

procedure plot(x,y: integer);     ( PLOT(x,y);                   )
 begin
     inline    ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBEA)            ( DEFW #BBEA 'PLOT'            )
 end;

procedure plotr(x,y: integer);    ( PLOTR(dx,dy);                )
 begin
     inline    ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBED)            ( DEFW #BBED 'PLOTR'           )
 end;

procedure draw(x,y: integer);     ( DRAW(x,y);                   )
 begin
     inline    ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBF6)            ( DEFW #BBF6 'DRAW'            )
 end;

procedure drawr(x,y: integer);    ( DRAWR(dx,dy);                )
 begin
     inline    ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B 'FIRMWARE'        )
                $BBF9)            ( DEFW #BBF9 'DRAWR'           )
 end;

procedure tag;                    ( pas d'argument               )
 begin
     inline    ($3E/1/            ( LD A,1                       )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BB63)            ( DEFW #BB63 'TAG'             )
 end;

procedure tagoff;                 ( pas d'argument               )
 begin
    inline     ($3E/O/            ( LD A,O                       )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BB63)            ( DEFW #BB63 'TAGOFF'          )
 end;

procedure origin(x,y: integer);   ( ORIGIN(x,y);                 )
 begin
    inline     ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BBC9)            ( DEFW #BBC9 'ORIGIN'          )
 end;

function test(x,y: integer): integer; ( TEST(x,y)                )
var
 tst:integer;
 begin
  begin
    inline     ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BBFO/            ( DEFW #BBFO 'TEST'            )
                $32/tst/          ( LD (tst),A                   )
                $AF/              ( XOR A                        )
                $32/tst+1)        ( LD (tst+1),A                 )
  end;
  test:=tst;
 end;

function testr(x,y: integer): integer;  ( TESTR(x,y)             )
var
 tstr: integer;
 begin
  begin
    inline     ($ED/$5B/X/        ( LD DE,(X)                    )
                $2A/Y/            ( LD HL,(Y)                    )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BBF3/            ( DEFW #BBF3 'TESTR'           )
                $32/tstr/         ( LD (tstr),A                  )
                $AF/              ( XOR A                        )
                $32/tstr+1)       ( LD (tstr+1),A                )
  end;
  testr:=tstr;
 end;

procedure clg;                    ( pas d'argument               )

 begin
   inline      ($CD/$BE9B/        ( CALL #BE9B                   )
                $BBDB)            ( DEFW #BBDB 'CLG'             )
 end;

procedure gwin(x1,x2,y1,y2: Integer); ( GWIN(d,g,h,b);           )
 begin
   inline      ($ED/$5B/X1/       ( LD DE,(X1)                   )
                $2A/X2/           ( LD HL,(X2)                   )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BBCF/            ( DEFW #BBCF 'GWIN WIDHT'      )
                $ED/$5B/Y1/       ( LD DE,(Y1)                   )
                $2A/Y2/           ( LD HL,(Y2)                   )
                $CD/$BE9B/        ( CALL #BE9B                   )
                $BBD2)            ( DEFW #BBD2 'GWIN HEIGH'      )
```

```
(*************************************************)
(* TOOLS3.PAS  Le 12 oct 1985 (c) LANGLOIS Michel *)
(*************************************************)

procedure setwindow(c: integer);  { SETWINDOW(n);              }
 begin
     inline    ($3A/C/            { LD A,(C)                   }
               $CD/$BE9B/         { CALL #BE9B                 }
               $BBB4)             { DEFW #BBB4 'SET STREAM'    }
 end;

procedure window(c,x1,y1,x2,y2: integer);

begin                       { WINDOW(n,d,g,h,b);               }
      setwindow(c);

     inline    ($3A/X1/           { LD A,(X1)                  }
               $67/               { LD H,A                     }
               $3A/Y1/            { LD A,(Y1)                  }
               $57/               { LD D,A                     }
               $3A/X2/            { LD A,(X2)                  }
               $6F/               { LD L,A                     }
               $3A/Y2/            { LD A,(Y2)                  }
               $5F/               { LD E,A                     }
               $CD/$BE9B/         { CALL #BE9B                 }
               $BB66)             { DEFW #BB66 'SET FENET'     }
 end;

procedure swapwindow(c1,c2: integer); { SWAPWINDOW(w1,w2);}
 begin
     inline    ($3A/C1/           { LD A,(C1)                  }
               $47/               { LD B,A                     }
               $3A/C2/            { LD A,(C2)                  }
               $4F/               { LD C,A                     }
               $CD/$BE9B/         { CALL #BE9B                 }
               $BBB7)             { DEFW #BBB7 'SWAP'          }
 end;

procedure pen(i: integer);        { PEN(n);                    }
 begin
     inline    ($3A/I/            { LD A,(I)                   }
               $CD/$BE9B/         { CALL #BE9B 'FIRMWARE'      }
               $BB90)             { DEFW #BB90 'PEN'           }
 end;

procedure paper(i: integer);      { PAPER(n);                  }
 begin
     inline    ($3A/I/            { LD A,(I)                   }
               $CD/$BE9B/         { CALL #BE9B 'FIRMWARE'      }
               $BB96)             { DEFW #BB96 'PAPER'         }
 end;

(*********************************************************)
(*  TOOLS4.PAS                                           *)
(*  SPECROND adapte par LANGLOIS Michel   18 oct 1985    *)
(*  (c) JR Hibberline    in    ELECTRONICS & COMPUTING   *)
(*                              FEBRUARY 1985            *)
(*********************************************************)

procedure cercle(cx,cy,r: integer);
var
  e,x,y: integer;
 begin
  e:=3-2*r;     { e : terme d'erreur pour le calcul d'un point }
  x:=r; y:=0;
    while y<x do
      begin
        plot(cx+x,cy+y); plot(cx+x,cy-y);
        plot(cx-x,cy-y); plot(cx-x,cy+y);
        plot(cx+y,cy+x); plot(cx+y,cy-x);
        plot(cx-y,cy-x); plot(cx-y,cy+x);
      if e<0 then e:=e+4*y+6 else
      begin
       e:=e+4*(y-x)+10; x:=x-1;
      end;
        y:=y+1;
    end;
end;
```

```
(*******************************************************************)
(***  TOOLS5.PAS    20 NOV 1985                           ********)
(***  LANGLOIS (c) 1985   ----   SINUS ET COSINUS EN ENTIER ********)
(*******************************************************************)

const
sintab: array[1..91] of integer
         =(0,175,349,523,698,872,1045,1219,1392,1564,1736,
           1908,2079,2250,2419,2588,2756,2924,3090,3256,3420,
           3584,3746,3907,4067,4226,4384,4540,4695,4848,5000,
           5150,5299,5446,5592,5736,5878,6018,6157,6293,6428,
           6561,6691,6820,6947,7071,7193,7314,7431,7547,7660,
           7771,7880,7986,8090,8192,8290,8387,8480,8572,8660,
           8746,8829,8910,8988,9063,9135,9205,9272,9336,9397,
           9455,9511,9563,9613,9659,9703,9744,9781,9816,9848,
           9877,9903,9925,9945,9962,9976,9986,9994,9998,10000);

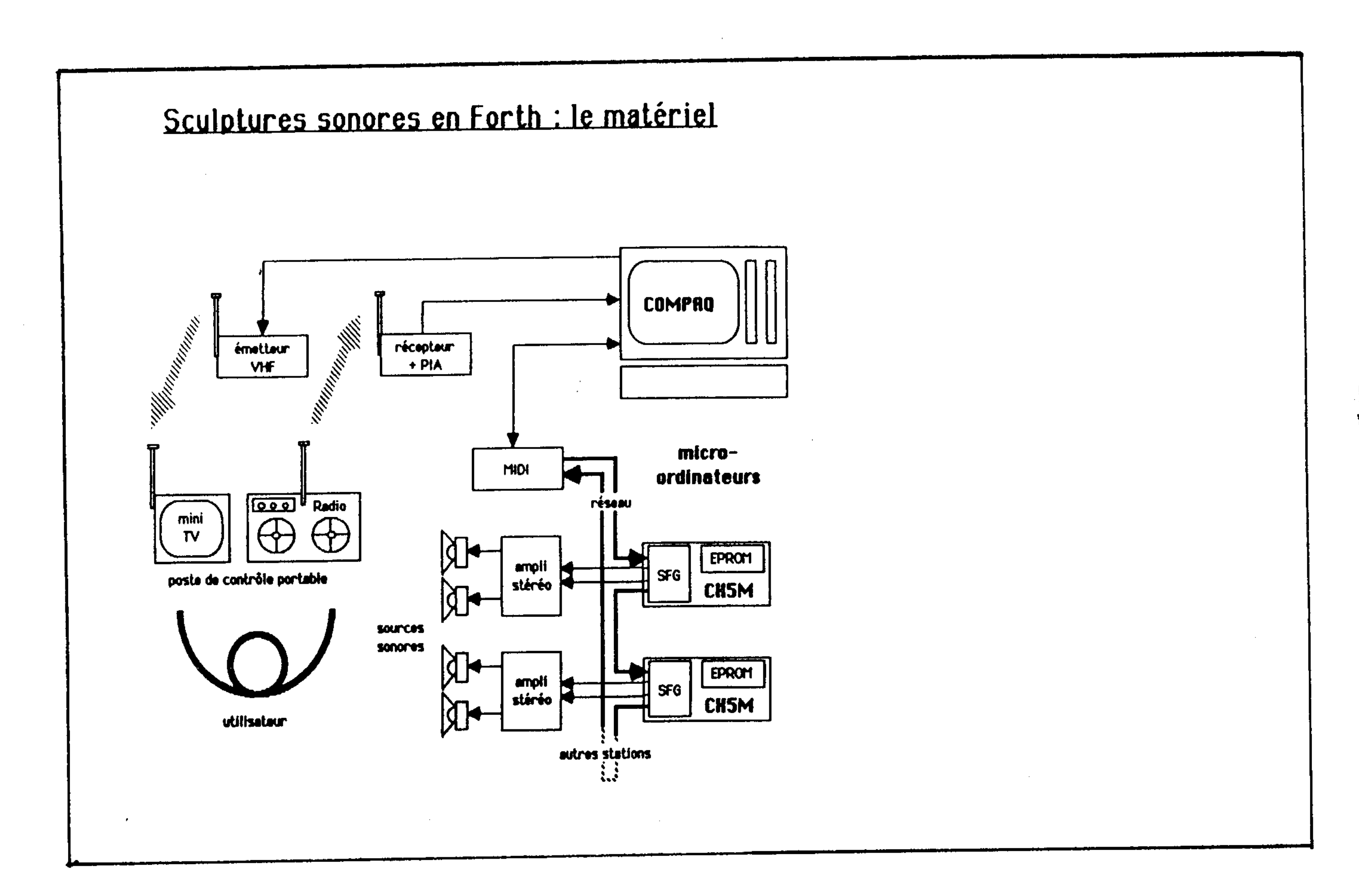
function dsin(xx:integer):integer;
begin
if xx>90 then
  xx:=180-xx;
 dsin:=sintab[xx+1];
end;

function psin(xx:integer):integer;
begin
xx:=xx mod 360;
if xx<0 then xx:=xx+360;
if xx>180 then  begin xx:=xx-180; psin:=-(dsin(xx)); end
 else psin:=dsin(xx);
end;

function pcos(xx:integer):integer;
begin
  xx:=(xx mod 360)+90;
pcos:=psin(xx);
end;

function sine(a,b: integer):integer; { sin(a)*b }
var fp,tp,tb:real;
begin
  tb :=b;
  tp :=psin(a);
  fp :=(tp*b);
  sine:=round(fp/10000);
end;

function cose(a,b: integer):integer; { cos(a)*b }
var fp,tp,tb:real;
begin
  tb :=b;
  tp :=pcos(a);
  fp :=(tp*b);
  cose:=round(fp/10000);
end;
```



VOS QUESTIONS ... NOS REPONSES

Au fil des mois, vous avez pu éprouver quelque difficulté à comprendre ou à adapter tel ou tel programme à votre système. Si dans la majorité des cas, un simple appel téléphonique a pu vous aider, il n'en est pas toujours ainsi quand un point de détail s'avère un peu délicat. C'est pourquoi, les problèmes rencontrés qui nous seront soumis seront communiqués aux auteurs concernés et nous nous efforcerons en retour de publier rapidement la réponse.

NUMERO 16: UN DECOMPILATEUR RECURSIF: de nombreuses lettres désepérées nous demandent la définition de CLIT. La réponse est simple: si vous n'en disposez pas, omettez la partie de définition concernant ce mot. En effet, CLIT est similaire à LIT mais appliqué aux valeurs litérales huit bits. Ce mot est compilé par LITERAL. On le retrouve généralement sur le FORTH APPLE et ORIC. Par contre, il est inexistant sur THOMSON ou HECTOR (et les autres...).

NUMERO 20: VIRGULE FLOTTANTE EN 83-STANDARD: une demande de traducteurs bénévoles figurait en fin d'article. Celle-ci a été satisfaite et concerne l'article ALGORITHME DE CORDIC qui précède. Merci quand même à ceux qui ont proposé leur aide. Nous ne les oublions pas, nous avons d'autres articles à faire traduire.